Register No: 21L31A0585 Experiment No: 01 Date:

S. No	Component	Max. Marks	Marks Secured
1	Preparedness	2	
2	Viva-Voce	2	
3	Experiment	3	
4	Analysis & Record	3	
Total		10	
Date		Signature of the Lab teacher	

AIM: a) Write a program to implement dynamic arrays.

Description :- Dynamic memory allocation can be defined as a procedure in which size of a datastructure is changed during the runtime. This can be done by using 4 methods.

- malloc()
- calloc()
- realloc()
- free()

malloc :- "memory allocation" is used to dynamically allocate a single block of memory with a specified size.

Syntax - (cast-type*) malloc (n * sizeof (datatype));

calloc :- "contiguous allocation" is used to dynamically allocate the specified number of blocks of memory of a specified type.

Syntax - (cast-type*) calloc (n, sizeof (data-type));

Register No : Experiment No : Date:

Realloc () :- "re-allocation" is used to change the memory allocated dynamically of a previously allocated memory.

Syntax :- ptr = (int*) calloc (n, sizeof (int));
ptr = realloc (ptr, n * sizeof (int));

free () :- "free" is used to dynamically de-allocate the memory. It helps to reduce wastage of memory by freeing it.

Syntax :- free (ptr);

Program

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
  int n, i, * ptr, sum = 0;
  printf ("Enter number of elements");
  scanf ("%d", &n);
  ptr = (int *) calloc (n, sizeof (int));
  if (ptr == NULL)
  {
    printf ("unable to allocate memory\n");
    exit (0);
  }
```

Register No : Experiment No : Date:

```
else
{
    printf("memory allocated using calloc");
    printf("Enter elements in array :\n");
    for(i=0; i<n; ++i)
    {
        scanf("%d", ptr+i);
        sum = sum + *(ptr+i);
    }
    printf("sum = %d", sum);
}
free(ptr);
return 0;
}
```

output

Enter number of elements : 5
Memory allocated using calloc.
Enter elements in array :
1 2 3 4 5
sum = 15.

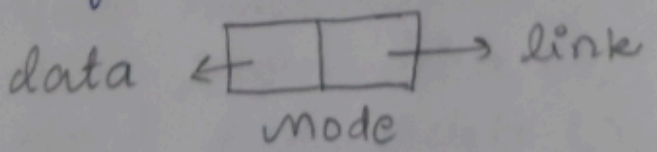
Register No: 2117140585 Experiment No: 02. Date:

S. No	Component	Max. Marks	Marks Secured
1	Preparedness	2	
2	Viva-Voce	2	
3	Experiment	3	
4	Analysis & Record	3	
Total		10	
Date		Signature of the Lab teacher	

AIM: Write a program to implement a single linked list and its operations.

Description

Singly linked list is a linear data structure. It is defined as the collection of ordered set of elements. A node in the singly linked list consists of 2 parts- data part and link part. Data part of the node stores actual information that is to be represented by the node while the link part of the node stores the address of its next successor.



Operations in single linked list:-

- Insertion
- Deletion
- Searching
- Displaying.

Register No : | Experiment No : | Date:

- Insertion can be done in 3 ways -
  - → Insertion at beginning
  - → Insertion at ending
  - → Insertion at specified position
- Deletion can be done in 3 ways -
  - → Deletion at begining
  - → Deletion at ending
  - → Deletion at specified position

Program

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
  int data;
  struct node *next;
};
struct node * head;
void beginsert ();
void lastinsert ();
void randominsert ();
void begin_delete ();
void last_delete ();
void random_delete ();
void display ();
void search ();
```

Register No : Experiment No : Date:

```
void main ()
{
  int choice = 0;
  while (choice != 9)
   {
     printf("choose one option from the list");
     printf("\n 1. Insert in begining
             \n 2. Insert at last
             \n 3. Insert at random location
             \n 4. Delete from beginning
             \n 5. Delete from last
             \n 6. Delete after specified location
             \n 7. Search for an element
             \n 8. Show
             \n 9. Exit \n");
  printf ("-Enter your choice");
  scanf (" %d", &choice);
  switch (choice)
  {
     case 1:
     beginsert();
     break;
    case 2:
     lastinsert();
     break;
```

Register No : Experiment No : Date:

```
case 3:
    randominsert();
    break;
case 4:
    begin_delete();
    break;
case 5:
    last_delete();
    break;
case 6:
    random-delete();
    break;
case 7:
    search();
    break();
case 8:
    display();
    break();
case 9:
    exit(0);
    break;
default:
    printf("Please enter valid choice");
}
}
}
```

Register No : Experiment No : Date:

```
void beginsert ()
{
 struct node *ptr;
 int item;
 ptr = (struct node *) malloc (sizeof(structnode));
 if (ptr == NULL)
  {
   printf ("overflow");
  }
 else
  {
   printf(" Enter value \n");
   scanf (" %d ", &item);
   ptr -> data = item;
   ptr -> next = head;
   head = ptr;
   printf (" \n node inserted");
  }
}
void lastinsert ()
{
  struct node *ptr, *temp;
  int item;
  ptr = (struct node*) malloc (sizeof (structnode));
  if (ptr == NULL)
  { printf ("\n overflow");
  }
```

Register No : Experiment No : Date:

```
else
{
printf("Enter value");
scanf("%d, &item);
ptr -> data = item;
if (head == NULL)
{
    ptr -> next = NULL;
    head = ptr;
    plintf("node inserted");
}
else
{
   temp = head;
   while (temp -> next != NULL)
   {
    temp = temp -> next;
   }
  temp -> next = ptr;
  ptr -> next = NULL;
 printf("node inserted");
}
}
}
void randominsert()
{ int loc, item, i;
  struct node *ptr, *temp;
  ptr = malloc(sizeof(structnode));
```

Register No : Experiment No : Date:

```
if(ptr == NULL) {
printf (" overflow ");
} else
{
printf (" enter element");
scanf ("%d", & item);
ptr → data = item;
printf (" Enter location");
scanf ("%d", & loc);
temp = head;
for (i=0; i<loc; i++)
{ temp = temp→next;
  if (temp == NULL)
  { printf ("cant insert");
    return;
  }
}
ptr→ next= temp→next;
temp → next = ptr;
printf (" node inserted");
}
}
void begin_delete()
{
struct node * ptr;
if (head == NULL)
{ printf ("List empty");
}
else
{ ptr = head;
  head = ptr→next;
  free (ptr);
  printf (" node deleted at
    beginning");
}
void last_delete()
{
struct node * ptr, * ptr1;
if (head == NULL) {
  printf (" list empty"); }
else if (head → next == NULL)
{ head = NULL;
  free (head);
  printf (" node deleted ");
}
else {
ptr = head;
while ( ptr → next != NULL)
{ ptr1 = ptr;
  ptr = ptr → next;
}
ptr1 → next = NULL;
free (ptr);
printf (" Node deleted at last");
}}
void random_delete ()
{ struct node * ptr, * ptr1;
int loc, i;
printf (" enter location");
scanf (" %d", & loc);
ptr = head;
for (i=0; i<loc; i++)
{ ptr1 = ptr;
  ptr = ptr → next;
  if (ptr == NULL)
  { printf(" Cant delete");
    return;
  }
}
ptr1 → next = ptr → next;
free(ptr);
printf ("Deleted Node %d", loc+1);
}
```

Register No : Experiment No : Date:

```
void search ()
{
struct node * ptr;
int item, i = 0, flag;
ptr = head;
if ( ptr = = NULL)
{
printf(" empty list");
}
else
{
printf ("Enter item to
be searched");
scanf ("%d", & item);
while (ptr ! = NULL)
{ if (ptr→ data = = item)
{
printf ("item found at
location %d", i+1);
flag = 0;
}
else
{
  flag = 1;
}
i++;
ptr = ptr → next;
}
if (flag = = 1)
{ printf (" not found");
}
}
}
```

```
void display ()
{
struct node * ptr;
ptr = head;
if (ptr = = NULL)
{
  printf("nothing to
    print");
}
else
{
  printf ("printing
  values are");
  while (ptr != NULL)
  {
  printf ("%d", ptr →
                data);
    ptr = ptr → next;
  }
}
}
```

output :-

Choose one option from the list.
1. Insert at beginning
2. Insert at last
3. Insert at random location
4. Delete from beginning
5. Delete from ending
6. Delete after specified location
7. Search for an element
8. Show
9. Exit

- Enter your choice : 1
  Enter value : 5
  Node inserted.
- Enter your choice : 8
  printing values are .....
  5
- Enter your choice : 2
  Enter value : 8
  Node inserted.
- Enter your choice : 8
  printing values are .....
  5 8
- Enter your choice : 3
  Enter element : 2
  Enter location : 0
  Node inserted
- Enter your choice : 8
  printing values are ...
  5 2 8
- Enter your choice : 7
  Enter item to be searched : 5
  Item found at location